

Games from Basic Data Structures

Mara Bovee
University of New England
bovee.mara@gmail.com

Kyle Burke
Plymouth State University
kgburke@plymouth.edu

Craig Tennenhouse
University of New England
ctennenhouse@une.edu

May 23, 2016

Abstract

In this paper, we consider combinatorial game rulesets based on data structures normally covered in an undergraduate Computer Science Data Structures course: arrays, stacks, queues, priority queues, sets, linked lists, and binary trees. We describe many rulesets as well as computational and mathematical properties about them. Two of the rulesets, TOWER NIM and MYOPIC COL, are new. We show polynomial-time solutions to TOWER NIM and to MYOPIC COL on paths.

1 Introduction

1.1 Notation

- For an integer x we will use x^- to denote an integer less than or equal to x , and x^+ an integer at least as large as x .
- We use the \oplus operator to refer to the bitwise binary XOR operation. For example, $3 \oplus 5 = 6$.
- Other notation common to combinatorial game theory is mentioned below.

1.2 Combinatorial Game Theory

Two-player games with alternating turns, perfect information, and no random elements are known as *combinatorial games*. Combinatorial Game Theory concerns determining which of the players has a winning move from any *position* (game state). Some important considerations here follow.

- This paper only considers games under *normal play* conditions, meaning that the last player to make a move wins. (If a player can't make a legal move on their turn, they lose.)
- The sum of two game positions is also a game where a legal move consists of moving on one of the two position summands. The value of a game sum is simply the sum of the values of the summands.
- Many of the sections require the reader to be moderately comfortable with number values, nimbers (or Grundy values), and outcome classes (specifically, \mathcal{N} and \mathcal{P}).
- Understanding of the game-set notation (e.g. $\{*, 1|*, 2\}$) is necessary for understanding Lemma 7.2.

The authors recommend [2] and [4] for background texts on combinatorial game theory.

1.3 Algorithmic Combinatorial Game Theory

Algorithmic combinatorial game theory concerns the computational difficulty in determining the winnability, outcome class, or value of game positions. Given a position, how long does it take for an algorithm to determine whether the next player has a winning strategy? For many rulesets, this can be solved in time polynomial in the size of the representation of the position. These “easy” rulesets usually allow some sort of closed formula to determine the winner.

For others, the problem is known to be intractable. For example, solving SNORT is known to be PSPACE-complete[10], meaning that unless an efficient algorithm is found to solve all other problems in PSPACE, no polynomial-time algorithm can determine the winnability of all SNORT positions. For these “hard” games, usually the majority of the induced game tree must be inspected in order to determine the winner.

In algorithmic combinatorial game theory, a ruleset is considered *solved* if either the polynomial time “trick” is known, or if it has been shown to be intractable. Many rulesets remain unsolved: it is unknown whether they can be evaluated efficiently or not.

In this paper, we discuss known algorithmic results about each ruleset.

1.4 Data Structures

Data structures are logical structures used to store digital data in computing systems. Different structures facilitate different operations. For example, a max-heap can quickly add and remove elements, but is restricted to removing only the greatest element.

A data structures course in computer science is a common course in the basic programming sequence in many undergraduate curricula. The topics cover many logical structures to store digital data. This paper covers games based on: arrays, sets, stacks, queues, priority queues, linked lists, and binary trees. We define each of these in the following sections.

1.5 Organization

The following 7 sections are each devoted to a ruleset based on a particular data structure. We provide Table 1 as a handy key to corresponding data structures, ruleset names, and section numbers.

Data Structure	Ruleset	Section
Array	NIM	2
Sets	ANTONIM	3
Stacks	TOWER NIM	4
Queues	ROTISSERIE NIM	5
Priority Queues	GREEDY NIM	6
Linked Lists	MYOPIC COL	7
Binary Trees	MYOPIC COL	8
Graphs	many games	9

Table 1: Corresponding data structures, rulesets, and sections.

2 Arrays

An array is a linear data structure addressable by the natural numbers. Each array element is stored in a cell so that the first n elements have respective addresses $0, 1, \dots, n - 1$.

2.1 Ruleset: Nim

NIM is a simple combinatorial game that has been well-studied[5].

Definition 2.1 (Nim). NIM is an impartial combinatorial game where each position is represented by a list of natural numbers: $K = (k_0, k_1, \dots, k_{n-1})$, $k_i \in \mathbb{N}$. The position represented by $M = (m_0, m_1, \dots, m_{n-1})$ is an option of K if $\exists i \in 0, 1, \dots, n-1$:

- $\forall j \neq i : k_j = m_j$, and
- $m_i < k_i$

The Grundy value of any NIM position can be found in linear time: $\mathcal{G}(k_0, k_1, \dots, k_{n-1}) = k_0 \oplus k_1 \oplus \dots \oplus k_{n-1}$. There is a winning move from position K exactly when the Grundy value, $\mathcal{G}(K)$, is not zero.[5]

3 Sets

A set is a data structure where there is no implicit order on the elements, and copies of elements are not recorded, (each element appears at most once).

3.1 Ruleset: Antonim

Instead of keeping a list (or bag) of heaps as in NIM, we employ a set. In the resulting ANTONIM, each turn consists of choosing a heap from the set, removing some number of sticks from that heap, then replacing the original heap with the new heap. (If the resulting heap has no sticks, we don't bother to add it back.) The catch here is that each heap is simply a natural number, so if there is already a heap of that size in the set, the new heap is forgotten about; that player might as well have taken all the sticks in that heap. We provide a formal description in Definition 3.1.

Definition 3.1 (Antonim). ANTONIM is an impartial combinatorial game where each position is represented by $S \subset \mathbb{N} \setminus \{0\}$. The position represented by S' is an option of S if:

- $\exists x \notin S'$ such that $S = S' \cup \{x\}$, or
- $\exists x \in S, y \in S'$ such that $x \notin S'$ and $y < x$

Finding the outcome class for ANTONIM has already been solved for positions with up to three piles [4]. These rules are as follows:

- $\{\}$ is a terminal position; so the position is in \mathcal{P} .
- One pile: $\{k\} \in \mathcal{N}$, as there is always a winning move (by taking all sticks).
- Two piles: $\{a, b\} \in \begin{cases} \mathcal{P}, & \{a, b\} = \{2k+1, 2k+2\} \text{ for some } k \in \mathbb{N} \\ \mathcal{N}, & \text{otherwise} \end{cases}$
- Three piles: $\{a, b, c\} \in \begin{cases} \mathcal{P}, & (a+1) \oplus (b+1) \oplus (c+1) = 0 \\ \mathcal{N}, & \text{otherwise} \end{cases}$

This last case can be proven inductively or by a reduction to standard Nim. A winning move is to add one stone to each heap, make an appropriate move in Nim, then remove a stone from each heap. The appropriate Nim move will not eliminate any heap entirely since that would require two equal-sized heaps to be present, which is not possible in Antonim. Play continues until the game is down to two heaps.

There is currently no known polynomial time algorithm for determine the outcome class for an Antonim position containing four or more heaps, and computer analysis has not yet shed light on a possible method.

4 Stacks

A stack is a data structure where elements are added and removed. Each remove operation pulls out the most recently-added element. We can represent this with a list where items are added and removed from the same end, called the top. The remove operation (called *pop*) then always fetches the most recently added (*pushed*) element.

Consider the following stack (the top is on the right-hand side): $(5, 2, 4)$. If we pop an element off the stack, $(5, 2)$ will remain. If we then push 1 followed by 6, followed by another, pop, the resulting stack will be $(5, 2, 1)$.

4.1 Ruleset: Tower Nim

TOWER NIM is another spin-off of NIM. The difference here is that the heaps are arranged linearly. A move consists of removing sticks from the top (or front) heap. If all sticks in one heap are removed, then that heap is removed from the stack. TOWER NIM is similar to END NIM [3] but with the added restriction that only one end is accessible. We formalize this in Definition 4.1.

Definition 4.1 (TOWER NIM). TOWER NIM is an impartial ruleset where positions are described by a list of non-zero natural numbers: $L = (a_0, a_1, \dots, a_{n-1}, a_n)$. The position represented by L' is an option of L if either

- $L' = (a_0, \dots, a_{n-1})$, or
- $\exists b \in \mathbb{N} \setminus \{0\}$ such that $b < a_n$ and $L' = (a_0, \dots, a_{n-1}, b)$.

After a handful of games, it becomes clear that the winnability of the game is often determined by whether the top heap has 1 stick.

Observation 4.2. For games with a stack of all ones, $(1, 1, \dots, 1)$, the number is the xor-sum of all heaps.

Lemma 4.3. *When only the top is non-one, the position is always in \mathcal{N} .*

Proof. Let $(1, 1, \dots, 1, x)$ represent a position, G , of TOWER NIM, where $x > 1$. There are two cases, based on the length of the list.

- If the position has an even number of heaps, then the next player can take $x - 1$ sticks, reducing the game to a zero position.
- If the position has an odd number of heaps, then the next player can take all x sticks in the top heap, reducing the number of sticks to an even amount, a zero position.

Since the current player always has a winning move, G must be in \mathcal{N} . □

Corollary 4.4. *Let $(\dots, x, 1, \dots, 1)$ represent a position, G , of TOWER NIM, where $x > 1$ and n is the number of 1-heaps above x . Then the number of G is the parity of n .*

Proof. After n turns, the current position must be represented by (\dots, x) , which is in \mathcal{N} . □

Lemma 4.5. *Let $(\dots, 1, x)$ represent a position, G , of TOWER NIM. There are two cases for the number of G :*

$x = 1$ Then either $G = 0$ or $G = *$.

$x \geq 2$ Then $G = *x$.

Proof. The first case is taken care of for us, using the previous corollary. The second case can be proven by strong induction on x . In the base case, G has both 0 and $*$ and options, by the previous corollary, and must equal $*2$. For the inductive case, when $x = k + 1$, it has all the children that $x = k$ has $(0, *, *2, \dots, *(k - 1))$, as well as $*k$, but does not include $*(k + 1)$. Thus, by the mex rule, $G = *(k + 1)$. □

Corollary 4.6. *Let (\dots, y, x) , where $y \geq 2$, represent a position, G , of TOWER NIM. There are two cases for the number of G :*

$x = 1$ Then $G = 0 (G \in \mathcal{P})$.

$x \geq 2$ Then $G \in \mathcal{N}$.

Proof. The first case occurs because the only option is a non-zero position as described in the previous lemma. The second case occurs because the first case is an option. \square

With these results, we can determine the outcome class of any position by counting the number of consecutive size-1 heaps on top of the stack and whether there is a bigger heap underneath the consecutive 1-heaps.

Corollary 4.7 (TOWER NIM is in P). *The outcome class of any TOWER NIM position can be determined in P.*

Proof. Consider a TOWER NIM position $G = (x_{n-1}, \dots, x_1, x_0)$. Then:

- if $\forall i : x_i = 1$ then, by Observation 4.2 $G = \begin{cases} 0 & (G \in \mathcal{P}), \quad n \text{ is even} \\ * & (G \in \mathcal{N}), \quad n \text{ is odd} \end{cases}$
- otherwise, let x_k be the non-one heap closest to the top. Now, by Corollary 4.4 $G \in \begin{cases} \mathcal{N}, & k \text{ is even} \\ \mathcal{P}, & k \text{ is odd} \end{cases}$

\square

5 Queues

Queues are another data structure very similar to stacks. The only difference is that the remove operation extracts the least-recently added element. We can represent this as a list: queues push elements to the *back* of the list and pop from the *front*, the opposite end.

Consider the following queue (the front is the left-hand side, the back is on the right): $(5, 2, 4)$. If we pop an element from the queue, $(2, 4)$ will remain. If we then push 1 followed by 6, followed by another, pop, the resulting queue will be $(4, 1, 6)$.

5.1 Ruleset: Rotisserie Nim

ROTISSERIE NIM is the queue-counterpart to TOWER NIM. In this game, the next player removes sticks from the heap in the front of the queue. Then, if there are any sticks left in that heap, it's queued back in to the back of the line. We formalize this in Definition 5.1. This is equivalent to ADJACENT NIM, which is itself a special case of VERTEX NIM played on a directed cycle[7].

Definition 5.1 (ROTISSERIE NIM). ROTISSERIE NIM is an impartial ruleset where positions are described by a list of non-zero natural numbers: $L = (a_0, a_1, \dots, a_n)$. The position represented by L' is an option of L if either

- $L' = (a_1, \dots, a_n)$, or
- $\exists b \in \mathbb{N} \setminus \{0\}$ such that $b < a_0$ and $L' = (a_1, \dots, a_n, b)$.

We begin by characterizing two heap games.

Theorem 5.2. *If $L = (a_0, a_1)$, then $L \in \mathcal{N}$ iff $a_0 > a_1$.*

Proof. First, assume $a_0 > a_1$. The next player can move to the position (a_1, a_1) , and continue to match the other player's moves until reaching the terminal position. Next, assume that $a_1 \geq a_0$. The only move from L is to a position of the form (a_1, a_0^-) . Since $a_0^- < a_0 \leq a_1$, this is an \mathcal{N} -position by the above argument, and therefore $L \in \mathcal{P}$. \square

The following theorem, proven in [7], completely characterizes the classes \mathcal{P} and \mathcal{N} where each heap has size at least two.

Theorem 5.3. *Let $L = (a_0, a_1, \dots, a_n), a_i \geq 2, \forall 0 \leq i \leq n$ be a position in ROTISSERIE NIM, and let $a_- = \min\{a_i\}_0^n$. Then $L \in \mathcal{N}$ iff n is odd or the smallest index j for which $a_j = a_-$ is even.*

We extend this to some small positions with heaps of size one.

Theorem 5.4. *If $a_0 = 1$ then $L \in \mathcal{P}$ if $a_1 > a_2$.*

Proof. Assume $a_1 > a_2$ and $a_0 = 1$. The only valid move is to (a_1, a_2) which is in \mathcal{N} by Theorem 5.2. \square

In order to complete the characterization of all three heap positions in ROTISSERIE NIM we must first prove the following lemma.

Lemma 5.5. *Let $L = (a_0, a_1, \dots, a_n)$ is a position in ROTISSERIE NIM. If $(a_1, \dots, a_n) \in \mathcal{N}$ then:*

- *If $(n+1)$ is odd and $L \in \mathcal{N}$ then $(a_1, \dots, a_n, 1) \in \mathcal{P}$, and*
- *If $(n+1)$ is even and $L \in \mathcal{N}$ then $(a_1, \dots, a_n, a_0 - 1) \in \mathcal{P}$.*

Proof. If L has an even number of heaps, then the same player who encounters heap a_0 will also encounter what remains of this heap on a subsequent turn. Therefore, if removing the entire heap does not move the game into a \mathcal{P} -position then there is no advantage to removing more than a single stick. Similarly, with an odd number of heaps there is no advantage to leaving more than a single stick for the other player to encounter on a subsequent turn. \square

Lemma 5.5 provides an interesting strategy, which leads to a win of if $L \in \mathcal{N}$. Using this strategy we prove another lemma necessary to complete our characterization of three heap games.

Lemma 5.6. *If $L = (a_0, a_1, \dots) \in \mathcal{P}$ then so is $L' = (a_0^-, a_1^+, a_2^-, a_3^+, \dots)$.*

Proof. Assume to the contrary that $L \in \mathcal{P}$ and $L' \in \mathcal{N}$. Therefore there is a move from L' to a \mathcal{P} -position $(a_1^+, a_2^-, a_3^+, \dots, a_0^{--})$. That means that a similar move can take L to $(a_1, a_2, a_3, \dots, a_0^{--})$ since $a_0^- \leq a_0$. In fact, each move to a position in \mathcal{P} from the starting position of L' can be copied by the first player in the starting position L , and hence $L \in \mathcal{N}$. This contradicts our assumption, hence $L' \in \mathcal{P}$ as well. \square

Theorem 5.7. *If $a_0 > 1$ then $L \in \mathcal{P}$ iff $a_1 > 1$ and $a_2 = 1$.*

Proof. Assume $L = (a_0, 2, 1)$. Since $(2, 1) \in \mathcal{N}$ by Theorem 5.2, Lemma 5.5 tells us that the first move should be to the position $(2, 1, 1)$. The second player can then move to $(1, 1)$, which is in \mathcal{P} . Therefore, $L \in \mathcal{P}$. By Lemma 5.6, $(a_0, a_1, 1) \in \mathcal{P}$ whenever $a_2 \geq 2$.

Next, assume $a_0, a_2 > 1$ and $a_1 > a_2$. Because $(a_1, a_2) \in \mathcal{N}$ by Theorem 5.2, by Lemma 5.5 the first player will move to position $(a_1, a_2, 1)$. Similarly, because $(a_2, a) \in \mathcal{N}$ the next player will move to $(a_2, 1, 1)$. The first player can then move to $(1, 1) \in \mathcal{P}$. Therefore, $(a_0, a_1, a_2) \in \mathcal{N}$ when $a_0, a_2 > 1$ and $a_1 > a_2$, and hence when $a_0 > 1$, the position $(a_0, a_1, a_2) \in \mathcal{P}$ iff $a_2 = 1$.

Finally, consider the case where $a_0 > 1$ and $a_1 < a_2$. Since $(a_1, a_2) \in \mathcal{P}$ by Theorem 5.2, $t(a_0, a_1, a_2) \in \mathcal{N}$. Therefore, if $a_0 > 1$ then $(a_0, a_1, a_2) \in \mathcal{P}$ iff $a_1 > 1$ and $a_2 = 1$. \square

6 Priority Queues

Priority queues are similar to stacks and queues, except that the order of elements does not necessarily determine the order they will be removed. When a remove-operation is called, the largest element is always removed.

Consider the following priority queue: $(5, 2, 4)$. If we remove an element from the queue, $(2, 4)$ will remain. If we then add 1 followed by 6, followed by another, remove, the resulting priority queue will be $(2, 4, 1)$.

6.1 Ruleset: Greedy Nim

GREEDY NIM is a ruleset just like NIM, except that players must choose from a heap with the greatest size to remove sticks from[4]. We give a formal version in Definition 6.1.

Definition 6.1 (Greedy Nim). GREEDY NIM is an impartial ruleset where positions are described by a multiset of non-zero natural numbers, M . M' is an option of M exactly when $\exists x \in M : \forall y \in M : x \geq y$ and $\#_M(x) = 1 + \#_{M'}(x)$ and either:

- $\forall a \in (M \cup M')$ either $x = a$ or $\#_M(a) = \#_{M'}(a)$, or
- $\exists b \in M \setminus \{x\} : \#_M(b) + 1 = \#_{M'}(b)$ and $\forall a(\neq b) \in (M \cup M')$ either $x = a$ or $\#_M(a) = \#_{M'}(a)$.

This game has already been solved; a polynomial-time algorithm exists to determine the outcome class of the game[1].

Theorem 6.2 (Greedy Nim). A GREEDY NIM position, G , is in \mathcal{P} exactly when there are an even number of heaps with the greatest size.

7 Linked Lists

A linked list is a data structure with more explicit structure than a stack, queue, or priority queue. Instead of describing which element gets removed, a linked list is a method for organizing elements. Indeed, stacks and queues can both be implemented using linked lists.

Each element in a linked list exists inside a list node, which also includes a reference to the next list node. (We consider only singly-linked lists, meaning each node does not have a reference to the previous node.) In other terms, a linked list is like a directed path graph, with a value at each vertex.

7.1 Ruleset: Myopic Col

There are many choices of rulesets that could be used on directed paths. In order to deviate from impartial games, we use a variant of COL, MYOPIC COL.

In COL, a turn consists of a player choosing an uncolored vertex and painting it their own color (blue or red). However, they not allowed to choose a vertex adjacent to another vertex with their color.

MYOPIC COL is played on a directed graph instead of undirected. When a player chooses a vertex, they are only restricted from choosing vertices with arcs pointing to neighboring vertices of their color. Arcs directed towards the chosen vertex do not need to be considered.

As with other games discussed, we provide a rigorous definition.

Definition 7.1 (Myopic Col). MYOPIC COL is a combinatorial game where positions are described by a directed graph, $G = (V, E)$, and a coloring of vertices either uncolored or painted red or blue, $(c : V \rightarrow \{\text{uncolored}, \text{red}, \text{blue}\})$. Each arc, $(a, b) \in E$ points from a to b . Another graph-coloring pair (G', c') is an option of (G, c) for player $A \in \{\text{red}, \text{blue}\}$ exactly when both

- $G' = G$, and
- $\exists v \in V :$
 - $\forall v' \in V \setminus \{v\} : c(v') = c'(v')$, and
 - $c(v) = \text{uncolored}$ and $c'(v) = A$ and $\nexists (v, b) \in E : c(b) = A$

The positions we consider in this section are on paths. ...

Consider a positions on a paths described as a list of the colors. For example, $(\text{blue}, \text{red}, \text{uncolored}, \text{uncolored}, \text{blue})$ represents path with five vertices with arcs going from-left-to-right.

Note that the game on a path with an arc from a colored vertex to another vertex is equivalent to the sum of two paths created by removing that arc. Thus, we must only consider situations where there is at most one colored vertex, and that vertex is at the end of the path.

Before we prove things about the Col paths, we need a general lemma about games with numeric and star values:

Lemma 7.2. $\forall x \in \mathbb{R} :$

$$x + * = \{x|x\} \quad (1)$$

$$x = \{x + *|x + *\} \quad (2)$$

Proof. First we prove (1):

$$\begin{aligned} \{x|x\} &= x \pm 0 && \text{(by the definition of switches)} \\ &= x + \{0|0\} \\ &= x + * \end{aligned}$$

Next we prove (2) by showing that $\{x + *|x + *\} - x = 0$. Notice that either player loses by playing on $\{x + *|x + *\}$, because $x + * - x = * \in \mathcal{N}$. However, since $-x$ is a number and $\{x + *|x + *\}$ is not, it is still better for both players to make that losing move. Thus, the sum is a loss for both players and equals 0. \square

Lemma 7.3. *Let G be a Myopic Col game played on a path with n uncolored vertices, possibly followed by a colored vertex, v . Then $G = \begin{cases} n \times *, & \text{no } v \\ (n-1) \times * - 1, & v \text{ is blue} \\ (n-1) \times * + 1, & v \text{ is red} \end{cases}$*

Proof. We proceed by strong induction on n :

Base Case: $n = 1$.

All three cases hold. In the first case, there is only one vertex that either can play on, so $G = *$, as the lemma states. In the second case, there is a single vertex only Right can color. Thus, $G = -1$. In the third case, $G = 1$ by the same logic.

Inductive Case: Assume $n > 1$ and both lemmas are true for all $0 < k < n$.

Consider G , the game played on a path with n uncolored vertices. The moves for Left can be derived from the inductive hypotheses. Starting by coloring the head, these are:

$$\begin{aligned} &(n-1) \times *, \\ &(0 \times * - 1) + (n-2) \times * = (n-2) \times * - 1, \\ &(1 \times * - 1) + (n-3) \times * = (n-2) \times * - 1, \\ &\vdots \\ &((n-3) \times * - 1) + 1 \times * = (n-2) \times * - 1, \\ &((n-2) \times * - 1) + 0 \times * = (n-2) \times * - 1 \end{aligned}$$

Except for the first one, these are all equal to $(n-2) \times * - 1$. Since $(n-2) \times * - 1 < (n-1) \times *$, they are all dominated by that first move. Thus, the best move for Left is to color the head of the path.

The same logic holds to show that Right's best move is also to color the head of the path. Thus, $G = \{(n-1) \times *|(n-1) \times *\} = n \times *$.

Now consider the case where G is a path of n uncolored vertices followed by a single blue vertex. Again consider the moves for Left, starting with playing at the head and continuing down the tail:

$$\begin{aligned} &(n-2) \times * - 1, \\ &(0 \times * - 1) + ((n-3) \times * - 1) = (n-3) \times * - 2, \\ &(1 \times * - 1) + ((n-4) \times * - 1) = (n-3) \times * - 2, \\ &\vdots, \\ &((n-4) \times * - 1) + (1 \times * - 1) = (n-3) \times * - 2, \\ &((n-3) \times * - 1) + (0 \times * - 1) = (n-3) \times * - 2 \end{aligned}$$

All but the first simplify to $(n-3) \times * - 2$, worse for Left than the first option, $(n-2) \times * - 1$.

Right has one more option:

$$\begin{aligned}
& (n-2) \times * - 1, \\
& (0 \times * + 1) + ((n-3) \times * - 1) = (n-3) \times *, \\
& (1 \times * + 1) + ((n-4) \times * - 1) = (n-3) \times *, \\
& \vdots, \\
& ((n-3) \times * + 1) + (0 \times * - 1) = (n-3) \times *, \\
& (n-2) \times * + 1
\end{aligned}$$

All but the first and last simplify to $(n-3) \times *$. $(n-2) \times * - 1 < (n-3) \times * < (n-2) \times * + 1$, so Right's best move is also to $(n-2) \times * - 1$.

Now,

$$\begin{aligned}
G &= \{(n-2) \times * - 1 | (n-2) \times * - 1\} \\
&= (n-2) \times * - 1 + * \\
&= (n-1) \times * - 1
\end{aligned}$$

The same steps can be used to show that a path ending in a red vertex has value $(n-1) \times * + 1$. □

Corollary 7.4. *On a path with n uncolored vertices, it is always optimal to color the head.*

Proof. For all of our cases in our analysis above, the best move was always to color the vertex at the head. □

Now we can evaluate any path graph (or collection of paths). The total value of the game is

Corollary 7.5. *For a graph that consists only of paths, G , the total value of M , the MYOPIC COL position on G equals $a \times * + b - c$, where:*

- *a is the number of uncolored vertices that either have no outgoing arc or have an outgoing arc that leads to another uncolored vertex.*
- *b is the number of uncolored vertices with their outgoing arc leading to a red vertex.*
- *c is the number of uncolored vertices with their outgoing arc leading to a blue vertex.*

Proof. We can break each path into sections by removing edges leaving colored vertices. By Lemma 7.3, this splitting does not change the values of the positions at all. Now the final formula, $a \times * + b - c$, is just the sum of all of the sums generated by using Lemma 7.3 on each piece. □

This formula is easy to evaluate, so we can solve this game efficiently.

Corollary 7.6 (MYOPIC COL on paths is in P). *The winnability of MYOPIC COL when played on path graphs can be determined in P.*

Proof. The formula from Corollary 7.5 can be evaluated in $O(n)$ time, where n is the number of vertices in the graph. □

In the next section, we put MYOPIC COL to work again, but on slightly more complex graphs. The computational complexity there is not yet known.

8 Binary Trees

In the same way that a linked list is a directed path graph, a binary tree is a directed tree graph with at most two outgoing arcs per vertex. Each node in the tree (each vertex) contains a value in the data structure, as well as up to two references to other nodes.

Binary trees are used to implement quickly-searchable structures on ordered data, as well as heaps, which can be used for fast priority queues.

8.1 Ruleset: Myopic Col on Trees

We can reuse the ruleset for MYOPIC COL from Section 7.1, and now consider the game played on binary trees. Unlike on paths, MYOPIC COL on binary trees has non integer/integer-plus-star values. Other dyadic rationals exist, such as $1/2$ and $-3/4$.

It seems natural that in a tree with an uncolored node as the root with two uncolored nodes as children of the root, coloring the root is the best move for both players, just as for a path that begins with two uncolored vertices.

Conjecture 8.1. *Consider a directed tree T_0 where the root vertex, r , is uncolored, and the child trees of r , T_1 and T_2 , are both non-null and have uncolored roots. For any tree T , let $G(T)$ be the MYOPIC COL position on T . Then $G(T_0) = * + G(T_1) + G(T_2)$.*

9 Graphs

Graphs covered in a data structures course are exactly the same as mathematical graphs with vertices, edges (or directed arcs), and values embedded in the vertices and/or edges. There are a wide host of rulesets on graphs (COL[6], CLOBBER, HACKENBUSH[4], NIMG[9], and more). Some games, such as UNDIRECTED VERTEX GEOGRAPHY, are in P[8]; others, such as SNORT, are PSPACE-complete[10].

10 Conclusions

This paper presents game rulesets that rely heavily on different common data structures. When relevant rulesets did not exist, new rulesets have been created: TOWER NIM, and MYOPIC COL.

We show polynomial-time algorithms that solve both TOWER NIM and MYOPIC COL on paths.

11 Future Work

Three open problems exist concerning the computational difficulty of games presented here: ROTISSERIE NIM, ANOTNIM, and MYOPIC COL on binary trees.

Although we have lots of results about ROTISSERIE NIM positions, we don't yet have an efficient algorithm for the general state.

Open Problem 11.1 (Rotisserie Nim). What is the computational complexity of ROTISSERIE NIM?

ANTONIM is a classic game that has resisted a solution. Recent work reduces the problem to a dynamic programming form[11]. Unfortunately, since integers can be represented with a logarithmic number of bits for even a small number of heaps, the size of the table can be exponential in the number of bits needed to describe the game. Therefore, it remains unknown whether a polynomial-time algorithm to determine the outcome class of an ANTONIM position exists.

Open Problem 11.2 (Antonim). What is the computational complexity of ANTONIM?

For MYOPIC COL, a polynomial-time solution exists on paths, but this does not immediately yield a solution on binary trees.

Open Problem 11.3 (Binary Tree Myopic Col). What is the computational complexity of MYOPIC COL played on binary trees?

References

- [1] M. H. Albert and R. J. Nowakowski. NIM restrictions. *Integers*, 4:G1, 10, 2004.
- [2] M. H. Albert, R. J. Nowakowski, and D. Wolfe. *Lessons in Play: An Introduction to Combinatorial Game Theory*. A. K. Peters, Wellesley, Massachusetts, 2007.
- [3] Michael H. Albert and Richard J. Nowakowski. The game of end-nim. *Electr. J. Comb.*, 8(2), 2001.

- [4] Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways for your Mathematical Plays*, volume 1. A K Peters, Wellesley, Massachusetts, 2001.
- [5] Charles L. Bouton. Nim, a game with a complete mathematical theory. *Annals of Mathematics*, 3(1/4):pp. 35–39, 1901.
- [6] John H. Conway. *On numbers and games (2. ed.)*. A K Peters, 2001.
- [7] Éric Duchêne and Gabriel Renault. Vertex nim played on graphs. *Theor. Comput. Sci.*, 516:20–27, 2014.
- [8] Aviezri S. Fraenkel, Edward R. Scheinerman, and Daniel Ullman. Undirected edge geography. *Theor. Comput. Sci.*, 112(2):371–381, 1993.
- [9] Masahiko Fukuyama. A nim game played on graphs. *Theor. Comput. Sci.*, 1-3(304):387–399, 2003.
- [10] Thomas J. Schaefer. On the complexity of some two-person perfect-information games. *Journal of Computer and System Sciences*, 16(2):185–225, 1978.
- [11] Zachary Silbernack and Robert Campbell. A winning strategy for the game of antonim. *arXiv.org Mathematics*, 2015.